DIGITAL ARCHITECTURE FOR RECONFIGURABLE COMPUTING IN DIGITAL SIGNAL PROCESSING

PRIORITY CLAIM

[1] This application claims priority from European patent application

5 No. 03425055.5, filed January 31, 2003, which is incorporated herein by reference.

FIELD OF THE INVENTION

[2] The present invention relates generally to an embedded architecture including a microcontroller and a memory device suitable for digital signal processing based on reconfigurable computing.

10 **[3]** The invention may find application in many environments such as in multimedia applications, audio and image compression and decompression standards, security applications such as code encryption and voice/image recognition, and telecommunications protocol management such as TCP/IP and UMTS.

15 **[4]** **BACKGROUND**As is well known by those skilled in this specific technical field, a classic architectural solution to achieve high elaboration performance when dealing with critical algorithmic kernels is to enhance a general purpose microcontroller with application-specific signal processors and peripherals for the most time-critical functions.

20 **[5]** As a matter of fact, in order to achieve sufficient volumes of transactions in presence of standards with a variable level of compliance, these platforms must often be over designed to cover the worst case of all requirements.

[6] A further more fine-grain solution provides for a reconfigurability at instruction-set level, also improving the ease of interfacing peripherals. Another solution,

25 developed by the Company Tensilica, offers a configurable processor "Xtensa" where instructions can be easily added at design time within the pipeline; see in this respect the article by R. E. Gonzales "Xtensa: a configurable and extensible processor" IEEE Micro, Volume: 20 Issue 2, March-April 2000.

[7]   However, the computational logic for new instructions is hardwired at design time with an ASIC-like flow, hence the processor can not be reconfigured after fabrication. This, although very successful, is still an application-specific solution with high non-recurrent engineering costs due to design and mask production.

5   [8]   An appealing alternative option is that of exploiting a Field Programmable Gate Array (FPGA) technology combining standard processors with embedded FPGA devices. This further solution allows to configure into the FPGA at deployment time exactly the required peripherals, exploiting temporal re-use by dynamically reconfiguring the instruction-set at run time based on the currently executed

10   algorithm.

[9]   This solution is disclosed in the U.S. Patent No. 5,956,518 to A. De Hon, E. Mirsky, J. Knight, F. Thomas, assigned to the Massachussets Institute of Technology and having title: "Intermediate-grain reconfigurable processing device".

[10]   The existing models for designing FPGA/processor interaction can be

15   grouped in two main categories:

-   the FPGA is a co-processor communicating with the main processor through a system bus or a specific I/O channel;

-   the FPGA is described as a function unit of the processor pipeline.

[11]   The first group includes the GARP processor, known from the article by T.

20   Callahan, J. Hauser, and J. Wawrzynek having title: "The Garp architecture and C compiler" IEEE Computer, 33(4) : 62–69, April 2000. A similar architecture is provided by the A-EPIC processor that is disclosed in the article by S. Palem and S. Talla having title: "Adaptive explicit parallel instruction computing", Proceedings of the fourth Australasian COmputer Architecture Conference (ACOAC), January 2001.

25   [12]   In both cases the FPGA is addressed via dedicated instructions, moving data explicitly to and from the processor. Control hardware is kept to a minimum, since no interlocks are needed to avoid hazards, but a significant overhead in clock cycles is required to implement communication.

[13]   Only when the number of cycles per execution of the FPGA is relatively high,

30   the communication overhead may be considered negligible.

[14]   In the commercial world, FPGA suppliers such as Altera Corporation offer digital architectures based on the US Patent No. 5,968,161 to T.J. Southgate, "FPGA

based configurable CPU additionally including second programmable section for implementation of custom hardware support".

[15] Other suppliers (Xilinx, Triscend) offer chips containing a processor embedded on the same silicon IC with embedded FPGA logic. See for instance the US Patent 6,467,009 to S.P. Winegarden et al., "Configurable Processor System Unit", assigned to Triscend Corporation.

[16] However, those chips are generally loosely coupled by a high speed dedicated bus, performing as two separate execution units rather than being merged in a single architectural entity. In this manner the FPGA does not have direct access to the processor memory subsystem, which is one of the strengths of academic approaches outlined above.

[17] In the second category (FPGA as a function unit) we find some disclosed architectures known as:

- "PRISC" by R. Razdan and M. Smith "A high-performance microarchitecture with hardware-programmable functional units" Proceedings of the 27th Annual International Symposium on Microarchitecture, November 1994;

- "Chimaera" by Z.A. Ye, A. Moshovos, S. Hauck, P. Banerjee "Chimaera: A High-Performance Architecture with Tightly-Coupled Reconfigurable Functional Unit" Proceedings of the 27th International Symposium on Computer Architecture, 2000 Page(s): 225 –235;

- "ConCISe" by B. Kastrup, A. Bink, and J. Hoogerbrugge "ConCISe: A compiler-driven CPLD-based instruction set accelerator" Proceedings of the Seventh Annual IEEE Symposium on Field-Programmable Custom Computing Machines, April 1999.

[18] In all these models, data are read and written directly on the processor register file minimizing overhead due to communication. In most cases, to minimize control logic and hazard handling and to fit in the processor pipeline stages, the FPGA is limited to combinatorial logic only, thus severely limiting the performance boost that can be achieved.

[19] Later attempts, like the "OneChip" solution by R. Wittig, and P. Chow "OneChip: An FPGA Processor With Reconfigurable Logic Proceedings" disclosed in

the IEEE Symposium on Field-Programmable Custom Computing Machines, pp.126-135, Napa Valley, California, March 1996, or the processor architectures proposed in the already cited US patents Nos. 5,956,5181 and 6,026,481 address the communication problem effectively sharing registers between a processor core and an independently embedded FPGA device.

[20]   These solutions represent a significant step toward a low-overhead interface between the two entities. Nevertheless, due to the granularity of FPGA operations and its hardware oriented structure, their approach is still very coarse-grained, reducing the possible resource usage parallelism and again including hardware issues not familiar nor friendly to software compilation tools and algorithm developers.

[21]   Thus, a relevant drawback in this approach is often the memory data access bottleneck that often forces long stalls on the FPGA device in order to fetch on the shared registers enough data to justify its activation.

[22]   A more recent architecture exploiting a remarkable trade-off between the models above cited is known as "Molen" processor, developed at the TUDelft; see in this respect the article: "The MOLEN rm-coded Processor", Proceedings of the 11th International Conference on Field-Programmable Logic and Applications 2001 (FPL2001), Belfast, Northern Ireland, UK, August 2001.

[23]   "Molen" main advantage is to utilize commercially available FPGA devices to build an embedded reconfigurable architecture that couples existing processor models (Altera Nios, IBM PowerPC) with well known gate-array technology (Altera Apex 20KE, Xilinx Virtex II Pro) obtaining significant performance speed-up for a broad range of DSP algorithms.

[24]   However, even this solution presents some drawbacks due to the fact that the extension to the processor instruction set are designed by the architecture designers, and "microcoded" in the architecture itself, rather than developed at compilation time by the user.

[25]   Moreover, due to the coarse grain of the tasks involved in the instruction set extension, the size of the introduced reconfigurable logic can severely affect the energy consumption for a given algorithm.

Summary of invention

[26]   An embodiment of the present invention is that of providing a new kind of embedded digital architecture having functional and structural features capable to offer significant performance and energy consumption enhancements with respect to a traditional signal processing device.

[27]   Moreover, such a new architecture should provide a programmer-friendly algorithm development environment based on C language while ensuring usual compilation and software evaluation tools.

[28]   An embodiment of the invention overcomes the limitations of similar preceding architectures by relying on a reconfigurable device of different nature, and a new approach to processor/gate-array interface. The proposed architecture exploits concurrent use of hardwired computational logic and of an additional data elaboration channel comprising a pipelined array of configurable, look-up table based cells controlled by a special purpose control unit.

[29]   According to a first embodiment of the invention, the digital architecture previously indicated comprises: a processor, structured to implement a Very Long Instruction Word elaboration mode by a general purpose hardwired computational logic, and an additional data elaboration channel comprising a reconfigurable function unit based on a pipelined array of configurable look-up table based cells controlled by a special purpose control unit, thus easing the elaboration of critical kernels algorithms.

[30]   Features and advantages of the digital architecture according to this invention will become apparent from the following description of a best mode for carrying out the invention given by way of non-limiting example with reference to the enclosed drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[31]   **FIG. 1** is a block diagram of a digital architecture for digital signal processing according to an embodiment of the present invention;

FIG. 2 is a block diagram of a hardware register locking mechanism of a register file incorporated into the architecture of **FIG.1** according to an embodiment of the invention;

FIG. 3 is a schematic view of a special configuration cache added to the architecture of **FIG. 1** according to an embodiment of the invention; and

FIG.4 is a block diagram of a detailed view of a particular gate array structure included in the architecture of **FIG. 1** according to an embodiment of the invention.

5 ## DETAILED DESCRIPTION

[32] With reference to the drawings views, generally shown at *1* is a digital embedded architecture realized according to an embodiment of the present invention for digital signal processing based on reconfigurable computing. The architecture *1* is schematically shown in **FIG. 1** and referred to hereinafter as

10 "XiRisc" (Extended instruction set RISC).

[33] The XiRisc architecture *1* comprises a processor *2* and a memory device *3* integrated into a same chip. The processor *2* is structured to implement a Very Long Instruction Word (VLIW) RISC architecture featuring two concurrent, pipelined data elaboration channels operating concurrently over a set of general purpose hardwired

15 function units specifically designed to improve its performance in a signal processing application environment.

[34] Target applications for the architecture *1* include a relevant collection of signal processing algorithms. For instance, the main environments where the proposed architecture *1* may offer valuable advantages are multimedia applications such as

20 audio and image compression and decompression standards, security applications such as code encryption and voice/image recognition, and telecommunications protocol management such as TCP/IP and UMTS.

[35] A detailed analysis of the cited algorithms could explain how the computational load that they require is not uniformly distributed. For instance, small

25 and very critical kernels, that can be described by few lines with high level programming languages, often require huge portions of overall processing time and energy, while all the many other functionalities composing the algorithm, albeit critical to its completion, become negligible in terms of resource utilization.

[36] The described embodiment of the present invention focuses on easing the

30 elaboration of such critical kernels, partitioning the computational load that they introduce on two concurrent elaboration cores: the first relying on a standard

software-based compilation on a standard digital processor core, and the second focused on hardware programmable logic, to be performed on the cell array.

[37]   The processor *2* is provided with an additional run time configurable data path, capable of introducing a potentially infinite number of virtual application specific function units.

[38]   The data path is tightly integrated in a processor core *3*, receiving inputs from a register file *4* and writing results on dedicated write back channels *5, 6* over the register file *4*. The architecture *1* is thus effectively based on three different and concurrent data elaboration flows, two of which are fed each cycle by instruction fetch and one based on an independent, variable latency pipeline implemented on the configurable datapath.

[39]   The instruction set extension is that of a reconfigurable architecture. The hardware extension is modeled as any other function unit of the processor *2*, fully controlled by assembly instructions that are "in lined" in the source code. Manual identification of the extracted computational kernels is provided too.

[40]   The main difference between the proposed embodiments and other previously described architectures is that the reconfigurable function unit (PiCoGa) is not a hardware-based FPGA device but rather a configurable data-path controlled by a dedicated control unit *8*.

[41]   All the known solution including just a hardware based embedded FPGA may have two possible drawbacks:

- FPGA usage and programming involves hardware development issues not usual to compilation tools and users proficient in high level languages;
- Up to now, FPGA-based units are either hardware based devices that involve a large silicon area, a relevant energy consumption and heavy reanalysis of the target algorithms in order to be executed on the array, or small units that are not capable to efficiently handle multi cycle latency instructions, and are often limited to combinatorial elaboration only, severely restricting the possible application domains.

[42]   However, a specific special purpose and configurable gate-array *7* has been advantageously provided to handle configurable pipelines of data. Hereinafter, we

will make reference to this specific array as "PiCo-Array", or Pipelined-Configurable-Array.

[43] Rather than being some kind of embedded, programmable hardware logic array, the PiCo-Array is a special function unit, comprising a pipelined array of lookup-based, configurable cells especially designed to virtually emulate a microprocessor data path.

[44] More specifically, topology of elaboration inside the PiCo-Array is not symmetrical in the two directions: cells are grouped in rows, each representing a possible stage of a customized pipeline.

[45] The PiCo-Array structure *7* can easily be represented by a control data flow graph, each row (or group of rows) corresponding to a different state.

[46] Differently from all existing configurable devices, the PiCo-Array pipeline activity is controlled by a dedicated control unit *8*. The unit *8* generates control signals for each row of the array *7*: every clock cycle only the rows involved in the computation phase to be executed in that specific cycle are activated, in a dataflow fashion.

[47] In order to support implementation of high level language constructions such as *while* and *for* loops, each array row can generate a feedback signal for the control unit *8*, that may represent a Boolean value from a condition check performed in the array core.

[48] For this reason, the PiCo-Array *7* not only supports multi-cycle functions, but it is explicitly structured to handle configurable pipelines, even deep ones.

[49] Each reconfiguration of the array may implement a peculiar data flow, with a number of stages suitable for the functionality to be performed that could even be unknown at compilation time. In fact, PiCo-Array latency instructions, as well as their results may depend on internal status informations stored in some cell register by previous instructions, thus dramatically reducing the required data throughput on the register file, that was a severe limitation to previous implementations.

[50] The following discussion considers the computational model adopted for an embodiment of the present invention.

[51] The XiRisc computational model takes advantage of strong synergy between different units tightly integrated in a single core.

[52]   In the prior art solutions, FPGAs behaving as co-processing or function units need to implement entire computational kernels to achieve results that justify the communication overhead due to their inclusion. As a consequence, when a specific task is composed of functions suitable to be mapped on a hardware-oriented device

5   and operators which could not be efficiently implemented in such devices, it has either to be completely executed on the processor core leaving the FPGA unused or to be entirely re-mapped on the array stalling for long periods processor execution.

[53]   On the contrary, the pipelined approach of the PiCo-Array makes the communication overhead with other function units as small as possible, thus allowing

10   to efficiently partition the operations composing a single task to the function unit that best fit them.

[54]   System control, memory and I/O interface, and general purpose arithmetics are performed on the two hardwired channels *5, 6*, whose VLIW configuration allows them to maintain a very high access rate to memory. Wide multipliers, variable

15   shifters, MACs which are so difficult to implement efficiently in FPGAs are executed on dedicated hardwired function units.

[55]   The configurable unit exploits parallelism of small portions of the task, implementing short-latency, hardware intensive application specific operators. In this way, utilization of the PiCo-Array *7* considerably increases, minimizing program flow

20   stalls and achieving a considerable parallelism between available resources, thus justifying its cost in terms of area for a wide range of applications.

[56]   The XiRisc *32*-slots register file features four read ports, that are used to support the issue of two RISC instructions each clock cycle. Two write ports are reserved for the two hardwired pipeline channels *5, 6*, while two other ports are

25   entirely dedicated to PiCo-Array *7* results write back, avoiding introduction of dedicated logic handling competition on the register file ports.

[57]   Given the unpredictable latency of PiCo-Array instructions *10*, Read-After-Write (RAW) hazards may occur on the destination registers of such instructions.

[58]   A hardware register locking mechanism, shown in **FIG. 2**, has been

30   introduced as the least intrusive way to handle such hazards. When a PiCo-Array instruction *10* is decoded its destination registers are locked, so that any following instruction trying to access them will cause a processor stall.

[59] Normal execution is restored only when the PiCo-Array completes the write-back operation **9**, unlocking its destination registers.

[60] For all kinds of reconfigurable architectures described so far, a very critical issue is the programming of the embedded configurable device. The reconfiguration of a gate array **7** can take hundreds to thousands of cycles, depending on the programmed region size.

[61] Even in cases when elaboration can in principle continue concurrently on other processor resources, the scheduling of the program flow will hardly be able to feed enough instructions to avoid stalls, that could overcome the benefits deriving from the use of the array. In the digital architecture according to an embodiment of the present invention, three different approaches may be adopted to overcome these limitations.

[62] Multi-context programming: According to an embodiment of the invention, the PiCo-Array **7** is provided with a first level cache, storing four configurations for each logic cell. Context switch takes only one clock cycle, providing four immediately available instructions.

[63] Region partitioning: According to an alternative embodiment of the invention, the array **7** is structured in blocks of dynamically variable size **12** (shown in **FIG. 4**). A block **12** can be reprogrammed while other blocks **12** are under execution, so that careful scheduling of operations and reconfigurations may result in no cache miss penalties even when the number of used configurations is large.

[64] Configuration cache: According to an alternative embodiment of the invention, in order to support complete concurrency between processor **2** execution and PiCo-Array **7** programming, a special configuration cache is added to the architecture **1**, featuring a dedicated very large (192-bit) bus to the array. Compression techniques are also used to reduce both size and time of configuration. The configuration process is triggered by the issue of a pGA-load instruction (see **FIG. 3**).

[65] All possible p-array operations are identified by a pGA-op code, whose length is not fixed (minimum six bits). Each pGA-load instruction contains information about:

A)   the configuration to be loaded into the array and its location in the configuration cache,

B)   the cache context (first level caching) and the block where the operation is to be downloaded. This information can be either stored in one of the processor registers *4*, that is referenced by bits [*25 .. 21*] of the instruction, or be expressed directly in bits [*25 .. 20*] as an immediate operand.

[66] A dedicated configuration handling logic *13* reads this information and controls the download mechanism. Processor elaboration will be stalled only if a pGA-op operation is issued requiring the same operation that is being configured, otherwise it will continue concurrently on the two hardwired channels *5, 6* or on a different block *12* or context of the array *7*.

[67] In any case, once the configuration has completed, elaboration will restart without altering program flow consistency. In case another pGA-load instruction is issued before the previous has finished its process (as introduced before, configuration lasts some hundred cycles depending on the required functionality) no stall occurs: the new pGA-load will be queued in a FIFO memory, and will be performed at the end of the current process.

[68] If the FIFO queue is full, elaboration is stalled. Only In case a pGA-op instructions refers to a functionality that is neither loaded nor scheduled on the queue is an "illegal op-code" exception raised.

[69] The architecture *1* may be part of an electronic system, such as a computer system.

[70] Advantages of the digital architecture according to the invention may be summarized in the following features:

Performance enhancement:

the reconfigurable unit allows for a very efficient implementation of software computational kernels, effectively decreasing the number of cycles required for the completion of a given algorithm, achieving speedup figures from 2x to 13x with respect to a standard programmable device.

11

Energy consumption minimization:

the mapping of computational cores on the reconfigurable unit allows for a significant minimization of the main sources of power consumption in programmable architectures, that is instruction fetch and access to the register file. Overall energy consumption figures up to 10% less with respect to a standard programmable device depending on the algorithm computed.

Familiar environment for high level languages programmers:

the pipelined structure of the PiCo-Array and the assembly-level granularity of tasks that are implemented on the array maintains a friendly environment for algorithmic development that does not necessarily involve hardware related skills.

Portability to high level software development tools:

the function unit model of the PiCo-Array/processor interface and the DFG-flow based pipeline control unit make the mapping of software kernels on the configurable unit straightforward for a standard C compiler, that can also take into account scheduling informations to enhance at compilation time instruction-level parallelism.

Very high resource parallelism:

the VLIW configuration, the presence of DSP specific hardwired function units and the processor/gate-array interface allows for a very high level of parallelism in computation. In particular, processor stalls due to interlocks and to the reconfiguration process are significantly less with respect to existing reconfigurable architectures.

Better overall performances over the prior art solutions:

in the prior art solutions no explicit pipeline handling device is embedded in the configurable device to handle data flow on a software-based fashion. In contrast, in an embodiment of the present invention the gate-array configuration is based on a logic synthesis approach, rather then using high-level languages constructs, that would be more familiar to the user and easier to be handled with high-level language like C.